Abysssec Research

1) Advisory information

Title	**: Novell iPrint Client Browser Plugin ExecuteRequest debug Parameter stack overflow**
Version	**: iPrint Client plugin v5.32 (XP SP3)**
Analysis	**: http://www.abysssec.com**
Vendor	**: http://www.novell.com**
Impact	**: Critical**
Contact	**: shahin [at] abysssec.com , info [at] abysssec.com**
Twitter	**: @abysssec**

2) Vulnerable version

Novell iPrint Client prior to v5.32

3) Vulnerability information

Class

1- Stack overflow

Impact

Successfully exploiting this issue allows remote attackers to execute arbitrary code or cause denial-of-service conditions on vulnerable version.

User interaction is required in order to open a malformed page.

Remotely Exploitable

Yes

Locally Exploitable

Yes

4) Vulnerabilities detail

The ExecuteRequest function of this activeX control send operations and parameters to the server without the need of page refresh. Here is how the function is called:

ExecuteRequest(Operation parameter, [ParameterType=value])

The above function send the operation parameter as first argument and the second argument is for parameters to the operation. For complete list of operations and parameter types refer to iPrint administration guide.

```
ExecuteRequest('op-client-interface-version', 'debug=value');
```

All the operation types need some special parameters except op-client-version-inf and op-client-interface-version so we use one of them for our tests because we obly are care about debug parameter.

As you see in the following code the ExecuteRequest function is implemented at address sub_10008770 as other functions like ShowMessageBox is implemented at sub_1000C300.

```
.rdata:10051780          dd offset aExecuterequest ; "ExecuteRequest"
.rdata:10051784          db    4
.rdata:10051785          db    0
.rdata:10051786          db    0
.rdata:10051787          db    0
.rdata:10051788          dd offset unk_1006450C
.rdata:1005178C          db    8
.rdata:1005178D          db    0
.rdata:1005178E          db    0
.rdata:1005178F          db    0
.rdata:10051790          dd offset sub_10008770
.rdata:10051794          align 10h
.rdata:100517A0          dd offset aShowmessagebox ; "ShowMessageBox"
.rdata:100517A4          db    5
.rdata:100517A5          db    0
.rdata:100517A6          db    0
.rdata:100517A7          db    0
.rdata:100517A8          dd offset unk_100644F8
```

```
.rdata:100517AC          db   3
.rdata:100517AD          db   0
.rdata:100517AE          db   0
.rdata:100517AF          db   0
.rdata:100517B0          dd offset sub_1000C300
```

The function process our string parameters and in case of validation perform the intended task for each special parameter. In part of that the debug parameter is checked and if exists then the function check for value of 'true' or 'yes'. And the result is program will be in some kind of debug mode operation. Here is the way these parameters are checked:

```
.text:10008863 loc_10008863:                    ; CODE XREF: sub_10008770+BEj
.text:10008863          lea    ecx, [ebp+Dst]
.text:10008869          push   ecx          ; Dst
.text:1000886A          push   ebx          ; int
.text:1000886B          push   offset aDebug   ; "debug"
.text:10008870          call   sub_1000F290
.text:10008875          add    esp, 0Ch
.text:10008878          cmp    eax, 0FFFFFFFFh
.text:1000887B          jz     loc_10008940
.text:10008881          lea    edx, [ebp+Dst]
.text:10008887          push   offset aTrue    ; "true"
.text:1000888C          push   edx          ; Str1
.text:1000888D          call   __mbsicmp
.text:10008892          add    esp, 8
.text:10008895          test   eax, eax
.text:10008897          jnz    short loc_100088B5
.text:10008899          lea    eax, [ebp+Dst]
.text:1000889F          push   offset aYes_0   ; "yes"
.text:100088A4          push   eax          ; Str1
.text:100088A5          call   __mbsicmp
.text:100088AA          add    esp, 8
.text:100088AD          test   eax, eax
.text:100088AF          jz     loc_10008940
```

At address 10008870 the parameter string is checked against 'debug' and the function returns -1 if not debug string. And in case of -1 the controls is transferred to loc_10008940 with a conditional jump. But if the parameter-type is debug it checks for true and yes after the '=' character in edx register by calling __mbsicmp function once for 'true' then for 'yes'.

The logic of the program act in a way that after these checks enters to a vulnerable loop that copies our string (for example 'debug=true') to the stack.

Here is the implementation of vulnerable loop:

```
.text:100088D9 loc_100088D9:                    ; CODE XREF: sub_10008770+1B2j
.text:100088D9              mov     ecx, [ebp+Str1]
.text:100088DC
.text:100088DC loc_100088DC:                    ; CODE XREF: sub_10008770+167j
.text:100088DC              cmp     al, 26h
.text:100088DE              jnz     short loc_100088E6
.text:100088E0              mov     byte ptr [ecx+edx], 0Ah
.text:100088E4              jmp     short loc_100088EB
.text:100088E6 ; ---------------------------------------------------------------------------
.text:100088E6
.text:100088E6 loc_100088E6:                    ; CODE XREF: sub_10008770+16Ej
.text:100088E6              mov     al, [edx]
.text:100088E8              mov     [ecx+edx], al
.text:100088EB
.text:100088EB loc_100088EB:                    ; CODE XREF: sub_10008770+174j
.text:100088EB              lea     edi, [ebp+var_214]
.text:100088F1              or      ecx, 0FFFFFFFFh
.text:100088F4              xor     eax, eax
.text:100088F6              lea     ebx, [ebp+var_1FF0]
.text:100088FC              repne scasb
.text:100088FE              not     ecx
.text:10008900              sub     edi, ecx
.text:10008902              mov     esi, edi
.text:10008904              mov     edi, ebx
.text:10008906              mov     ebx, ecx
.text:10008908              or      ecx, 0FFFFFFFFh
.text:1000890B              repne scasb
.text:1000890D              mov     ecx, ebx
.text:1000890F              dec     edi
.text:10008910              shr     ecx, 2
.text:10008913              rep movsd
.text:10008915              mov     al, [edx+1]
.text:10008918              mov     ecx, ebx
.text:1000891A              and     ecx, 3
.text:1000891D              inc     edx
.text:1000891E              test    al, al
.text:10008920              rep movsb
.text:10008922              jnz     short loc_100088D9
.text:10008924              mov     ebx, [ebp+Str]
```

But in case of large enough buffer instead of 'true' or 'yes' the problematic logic loop copies our string many times until it fill all of the stack with our arbitrary string. After overwriting the stack the program crashes and it happens when the SEH address is overwritten too. And the program execution flow will be transferred to our overwritten SEH address.

For example:

```
ExecuteRequest('op-client-interface-version', 'debug=AAAAAAAAAAAAA.....'); //A*1000
```

And about the length of buffer if it is too, the sub_1000F290 function when checking parameter type debug it also responsible for bound checking the entire argument. and if it return -1 we never reach our vulnerable loop target. Our examination showed that length of the second string argument should not exceeds more than 1024 character. (plus the 'debug=' string)

Patch analysis

In the later patched version of the software the vulnerable loop is replaced by a simple one. The new loop a counter at edi register is checked at the entry point against 0fffh so the max loop execution is 4096 times and it can copies one character to the stack in each iteration. But in the vulnerable more complex loop because of logic flow it copies our entire string parameters many times until it fill the entire stack.

```
loc_10008890:                   ; CODE XREF: sub_10008730+17Fj
.text:10008890          cmp     edi, 0FFFh
.text:10008896          jnb     short loc_100088B1
.text:10008898          mov     cl, [eax]
.text:1000889A          cmp     cl, 26h
.text:1000889D          jnz     short loc_100088A5
.text:1000889F          mov     byte ptr [edx+eax], 0Ah
.text:100088A3          jmp     short loc_100088A8
.text:100088A5 ; ---------------------------------------------------------------------------
.text:100088A5
.text:100088A5 loc_100088A5:                    ; CODE XREF: sub_10008730+16Dj
.text:100088A5          mov     [edx+eax], cl
.text:100088A8
.text:100088A8 loc_100088A8:                    ; CODE XREF: sub_10008730+173j
.text:100088A8          mov     cl, [eax+1]
.text:100088AB          inc     edi
.text:100088AC          inc     eax
.text:100088AD          test    cl, cl
.text:100088AF          jnz     short loc_10008890
```

Loop Patched version

Exploit

For the purpose of exploitation of this activeX control it is simple to find the offset f SEH address and transfer the control of program to our desired code, but because of possibility of using javascript and in turn allocating heap it is better to use the more reliable heap spray technic that is used in many of explorers and activeX exploits. The only point is not sending to many character more than 1024 character for the second argument of ExecuteRequest function and using 'debug= '+buffer.

```
op = "op-client-interface-version";
dbg = "debug=";
buffer= "AAAAAAAAA…."
 // maximum 1017 byte of buffer can contain SEH overwrite at a specific location or sprayed heap address.
target.ExecuteRequest (op, dbg+buffer);
```